QR-Decomposition Based Algorithms for Adaptive Volterra Filtering

Mushtaq A. Syed and V. John Mathews, *Senior Member, IEEE*

***Abstract*—A QR-RLS adaptive algorithm for nonlinear filtering is presented. The algorithm is based solely on Givens rotation. Hence the algorithm is numerically stable and highly amenable to parallel implementations. The computational complexity of the algorithm is comparable to that of the fast transversal Volterra filters. The algorithm is based on a truncated second-order Volterra series model; however, it can be easily extended to other types of polynomial nonlinearities. The algorithm is derived by transforming the nonlinear filtering problem into an equivalent multichannel linear filtering problem with a different number of coefficients in each channel. Such multichannel algorithms were not available in the past even for adaptive linear filtering applications. The derivation of the algorithm is based on a channel-decomposition strategy which involves processing the channels in a sequential fashion during each iteration. This avoids matrix processing and leads to a scalar implementation. Results of extensive experimental studies demonstrating the properties of the algorithm in finite and "infinite" precision environments are also presented. The results indicate that the algorithm retains the fast convergence behavior of the RLS Volterra filters and is numerically stable.**

I. Introduction

LINEAR filtering, mainly because of its analytical simplicity and its usefulness in a wide variety of applications, has progressed quite rapidly. However, there are a number of applications in which the performance of linear filters is unacceptable and one has to resort to nonlinear filters. For an introduction to the applications of nonlinear system models and approaches to adaptive nonlinear filtering, see [12]. In this paper we consider a numerically stable algorithm for exponentially weighted, recursive least squares adaptive nonlinear filters equipped with a truncated Volterra system model.

In the truncated Volterra series model the output $y(n)$ of any causal, discrete-time, time-invariant, nonlinear system is expressed as a function of the input $x(n)$ using the Volterra series expansion

$$y(n) = h_0 + \sum_{m_1=0}^{N_1-1} h_1(m_1)x(n-m_1) + \sum_{m_1=0}^{N_1-1}\sum_{m_2=0}^{N_2-1} h_2(m_1, m_2)x(n-m_1)x(n-m_2) + \cdots + \sum_{m_1=0}^{N_1-1}\cdots\sum_{m_p=0}^{N_p-1} h_p(m_1, m_2, \cdots, m_p)x \cdot (n-m_1)\cdots x(n-m_p) \quad (1)$$

where $h_r(m_1, \cdots, m_r)$ is the rth-order Volterra kernel [15], [16] of the system. Such models have found a variety of applications in the recent past, including equalization of satellite communication systems, nonlinear echo and noise cancellation problems, image processing, and several others. Several properties and applications of truncated Volterra system models are described in [12], [17]. One can assume, without loss of generality, that the Volterra kernels are symmetric, i.e., $h_r(m_1, \cdots, m_r)$ is left unchanged by any of the $r!$ permutations of the arguments $m_1, \cdots, m_r$. Early work on adaptive Volterra filters [7] were based on the LMS algorithm. Even though they are computationally simple, they suffer from slow and input signal-dependent convergence behavior and hence are not useful in many applications. More recently, Lee and Mathews [8] presented a fast transversal algorithm for recursive least squares (RLS) adaptive Volterra filtering. The fast RLS transversal Volterra filter is rapidly convergent; however, it suffers from poor numerical properties. Recently, the authors [20] have presented computational efficient and what appear to be numerically stable RLS adaptive lattice algorithms for nonlinear filtering.

In this paper we present another approach to the development of a fast and numerically stable RLS algorithm for adaptive nonlinear filtering using QR-decomposition of the data matrix. For ease of presentation, the nonlinearity is modeled using a second-order Volterra series expansion; however, the results can be easily extended to other polynomial system models. In our algorithm the QR-decomposition is performed using a sequence of Givens rotations [5]. The Givens rotation based QR-RLS algorithms have two outstanding features. 1) The algorithms are, in general, numerically stable because each operation is realized using numerically stable orthogonal transformations. 2) The algorithms exhibit a high degree of concurrency, regularity, and local interconnection. Hence, they are very good candidates for parallel implementation using array architectures, such as systolic arrays [10], [21]. QRD-based, fast RLS, single-channel algorithms for adaptive filtering have been presented by Cioffi [4], Bellanger [2], Regalia and Bellanger [14], Proudler *et al.* [13], and Ling [11]. Lattice based QR-RLS, multichannel adaptive filtering

Manuscript received April 7, 1992; revised March 4, 1993. This work was supported in part by the NSF under Grant MIP-8922146. Part of this paper was presented at the IEEE International Symposium on Circuits and Systems, San Diego, CA, May 1992. This paper was recommended by Associate Editor G. S. Moschytz.

M. A. Syed was with the Department of Electrical Engineering, University of Utah, Salt Lake City, UT 84112. He is now with Digicom Systems, Inc., Milpitas, CA 95035.

V. J. Mathews is with the Department of Electrical Engineering, University of Utah, Salt Lake City, UT 84112.

IEEE Log Number 9210021.

algorithms have been presented by Lewis [10] and Yang and Bohme [21]. Bellanger and Regalia [3] and Syed [19] have independently presented QR-RLS multichannel algorithms for adaptive filtering that are not based on the lattice structure.

The algorithm presented in this paper is derived by transforming the nonlinear filtering problem into an equivalent multichannel linear filtering problem. This multichannel linear filtering problem is different from the conventional multichannel problem considered in [3], [19] in the sense that the number of coefficients in each channel is different. The channels are processed individually in a sequential manner. This sequential processing avoids matrix processing and leads to a scalar implementation. The computational complexity of this sequential algorithm is comparable to that of the other fast RLS algorithms for Volterra filtering that are available in the literature. We also present results of extensive simulation studies demonstrating the properties of the algorithm. The algorithm appears to be numerically stable, at least for the signal configurations considered in this paper, and retains the fast convergence property that is a characteristic feature of RLS algorithms.

The rest of the paper is organized as follows. In Section II we present the derivation of the QR-RLS sequential algorithm for second-order Volterra filtering. The experimental results are presented in Section III. Finally, the concluding remarks are made in Section IV.

II. QR-RLS Adaptive Second-Order Volterra Filter

We consider the problem of adaptively minimizing the exponentially weighted recursive least squares cost function

$$\xi_N(n) = \sum_{k=1}^{n} \lambda^{n-k} \Bigg(d(k) - \sum_{m_1=0}^{N-1} \hat{a}_{m_1}(n) x(k-m_1) - \sum_{m_1=0}^{N-1} \sum_{m_2=m_1}^{N-1} \hat{b}_{m_1,m_2}(n) x(k-m_1) x(k-m_2) \Bigg)^2 \quad (2)$$

at each time, where λ $(0 < \lambda \leq 1)$ is a weighting factor that controls the speed of convergence and tracking capability of the algorithm, $d(k)$ and $x(k)$ are the desired response and the input signals, respectively, of the adaptive filter at time k, and $\hat{a}_{m_1}(n)$ and $\hat{b}_{m_1,m_2}(n)$ are the linear and quadratic coefficients, respectively, of the second-order adaptive Volterra filter at time n. (The upper limits of all three summations in the Volterra series expansion in (2) have been set equal only for convenience. The generalization to arbitrary limits is straightforward.) Let us define the input vector $\boldsymbol{X}(n)$ and the coefficient vector $\boldsymbol{W}(n)$, both of size $N(N+3)/2$ entries, at time n as

$$\boldsymbol{X}(n) = [x(n),\, x^2(n),\, x(n-1),\, x^2(n-1),\, x(n)x(n-1), \cdots, x(n)x(n-N+1)]^{\mathrm{T}}. \quad (3)$$

$$\boldsymbol{W}(n) = [\hat{a}_0(n)\, \hat{b}_{0,0}(n),\, \hat{a}_1(n), \cdots, \hat{b}_{0,N-1}(n)]^{\mathrm{T}} \quad (4)$$

where $(\cdot)^{\mathrm{T}}$ denotes the transpose of $(\cdot)$. Equation (2) can be rewritten using (3) and (4) as

$$\xi_N(n) = \sum_{k=0}^{n} \lambda^{n-k} (d(k) - \boldsymbol{W}^{\mathrm{T}}(n)\boldsymbol{X}(k))^2. \quad (5)$$

The optimal solution to the problem can be easily shown to be

$$\boldsymbol{W}_{\mathrm{opt}}(n) = \Omega^{-1}(n) P(n) \quad (6)$$

where

$$\Omega(n) = \sum_{k=0}^{n} \lambda^{n-k} \boldsymbol{X}(k) \boldsymbol{X}^{\mathrm{T}}(k) \quad (7)$$

and

$$P(n) = \sum_{k=0}^{n} \lambda^{n-k} d(k) \boldsymbol{X}(k). \quad (8)$$

Direct evaluation of (6) is, in general, computationally inefficient and prone to numerical instability.

In order to derive numerically stable and computationally efficient algorithms, we first establish a correspondence between the nonlinear filtering problem addressed here and linear, but multichannel filtering problems. This can be illustrated by rewriting the entries of the input vector $\boldsymbol{X}(n)$ as

$$\boldsymbol{X}_N(n) = \begin{bmatrix} x(n) & x(n-1) & \cdots & x(n-N+1) \\ x^2(n) & x^2(n-1) & \cdots & x^2(n-N+1) \\ & x(n)x(n-1) & \cdots & x(n-N+2)x(n-N+1) \\ & & \cdots & x(n-N+3)x(n-N+1) \\ & & \vdots & \vdots \\ & & \cdots & x(n)x(n-N+1) \end{bmatrix}. \quad (9)$$

Each row of the above data matrix may be thought of as made up of samples of a signal belonging to a different channel. Note that the number of samples that are used in the estimation process, and belonging to each of these channels, varies from channel to channel. There are $K = N+1$ channels and the signal $x_i(n)$ in the ith channel is defined as

$$x_i(n) = \begin{cases} x(n), & i = 1 \\ x(n)x(n-i+2), & i = 2, \cdots, N+1 \end{cases} \quad (10)$$

N_i, the number of samples from the ith channel that are used in the estimation process and is defined as

$$N_i = \begin{cases} N, & i = 1,\, 2 \\ N-i+2, & i = 3, \cdots, N+1. \end{cases} \quad (11)$$

Let us also define the total number of coefficients L as

$$L = \sum_{i=1}^{K=N+1} N_i. \quad (12)$$

It is convenient to reformulate the problem statement using the matrix notation. Let us define the error, desired signal, and input data vectors as

$$\underline{\boldsymbol{e}}(n) = [e(1), \cdots, e(n)]^{\mathrm{T}} \quad (13)$$

$$\underline{\boldsymbol{d}}(n) = [d(1), \cdots, d(n)]^{\mathrm{T}} \quad (14)$$

and

$$\underline{\boldsymbol{x}}_i(n) = [x_i(1), \cdots, x_i(n)]^{\mathrm{T}}, \qquad i = 1, \cdots, N+1. \quad (15)$$

Each vector contains n elements. The error vector $\boldsymbol{e}(n)$ resulting from approximating the desired vector $\boldsymbol{d}(n)$ can now be expressed as

$$\underline{\boldsymbol{e}}(n) = \underline{\boldsymbol{d}}(n) - X_L(n)\boldsymbol{W}(n) \quad (16)$$

where

$$X_L(n) = [\underline{\boldsymbol{x}}_1(n) \cdots \underline{\boldsymbol{x}}_1(n - N_1 + 1), \underline{\boldsymbol{x}}_2(n), \underline{\boldsymbol{x}}_2(n - N_2 + 1), \cdots, \underline{\boldsymbol{x}}_K(n - N_K + 1)] \quad (17)$$

and

$$\boldsymbol{W}(n) = [w_{1,1}(n), \cdots, w_{1,N_1}(n), \cdots, w_{K,N_K}(n)]^{\mathrm{T}}. \quad (18)$$

In the above, the subscript L denotes the number of columns of the data matrix $X_L(n)$. (In the general multichannel filtering problem, the ordering of the columns of $X_L(n)$ is as $[\underline{\boldsymbol{x}}_1(n), \cdots, \underline{\boldsymbol{x}}_1(n - (N_1 - N_2)), \underline{\boldsymbol{x}}_2(n)), \cdots, \underline{\boldsymbol{x}}_2(n - (N_2 - N_3)), \underline{\boldsymbol{x}}_3(n), \cdots, \underline{\boldsymbol{x}}_1(n - N_1), \cdots, \underline{\boldsymbol{x}}_K(n - N_K)]$. Here we have assumed without loss of generality that $N_1, \cdots, N_K$ are arranged in the descending order.) The adaptive filter tries to minimize the cost function which can be rewritten in matrix notation as

$$\begin{aligned}\xi(n) &= \boldsymbol{e}^{\mathrm{T}}(n)B^{\mathrm{T}/2}(n)B^{1/2}(n)\underline{\boldsymbol{e}}(n) \\ &= \|B^{1/2}(n)\underline{\boldsymbol{e}}(n)\|^2 \end{aligned} \quad (19)$$

where $\|(\cdot)\|$ denotes the Euclidean norm of $(\cdot)$ and $B^{1/2}(n)$ is the $n \times n$ element diagonal matrix given by

$$B^{1/2}(n) = \mathrm{diag}[\sqrt{\lambda^{n-1}}, \cdots, \sqrt{\lambda}, 1]. \quad (20)$$

In QR-decomposition techniques an orthogonal matrix is used to triangularize the weighted data matrix $B^{1/2}(n)X_L(n)$. Suppose that $Q_L(n)$ is an orthogonal matrix that triangularizes the weighted data matrix, i.e.,

$$Q_L(n)B^{1/2}(n)X_L(n) = \begin{bmatrix} R_L(n) \\ \mathbf{0} \end{bmatrix} \quad (21)$$

where $R_L(n)$ is an $L \times L$ element upper triangular matrix and $\mathbf{0}$ is a zero matrix of appropriate dimensions. In what follows, we use $\mathbf{0}$ to denote all zero matrices and vectors, irrespective of their dimensionality. Since orthogonal matrices are length preserving, the cost function can be expressed by

$$\begin{aligned}\xi(n) &= \|Q_L(n)B^{1/2}(n)\boldsymbol{e}(n)\|^2 = \left\| \begin{bmatrix} \underline{\boldsymbol{e}}_u(n) \\ \underline{\boldsymbol{e}}_v(n) \end{bmatrix} \right\|^2 \\ &= \left\| \begin{bmatrix} \boldsymbol{U}(n) \\ \boldsymbol{V}(n) \end{bmatrix} - \begin{bmatrix} R_L(n) \\ \mathbf{0} \end{bmatrix} \boldsymbol{W}(n) \right\|^2 \end{aligned} \quad (22)$$

where $\underline{\boldsymbol{e}}_u(n)$ and $\boldsymbol{U}(n)$ are $L \times 1$ element vectors and $\underline{\boldsymbol{e}}_v(n)$ and $\boldsymbol{V}(n)$ are $n - L \times 1$ element vectors. From (22) it follows that the LS coefficient vector $\boldsymbol{W}(n)$ can be computed by back substitution as

$$\boldsymbol{W}(n) = R_L^{-1}(n)\boldsymbol{U}(n). \quad (23)$$

It is easy to show that $Q_L(n)$ can be recursively updated as [6]

$$Q_L(n) = \hat{Q}_L(n) \begin{bmatrix} Q_L(n-1) & \mathbf{0} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix} \quad (24)$$

where $\hat{Q}_L(n)$ is the sequence of L rotations

$$\hat{Q}_L(n) = \hat{Q}_{(L)}(n) \cdots \hat{Q}_{(1)}(n) \quad (25)$$

such that

$$\hat{Q}_L(n) \begin{bmatrix} \sqrt{\lambda}R_L(n-1) \\ \mathbf{0} \\ \tilde{x}(n) \end{bmatrix} = \begin{bmatrix} R_L(n) \\ \mathbf{0} \\ \mathbf{0}^{\mathrm{T}} \end{bmatrix} \quad (26)$$

and $\tilde{x}(n)$ is the last row of the data matrix $X_L(n)$. The first rotation $\hat{Q}_{(1)}(n)$ in (25) annhilates $x_1(n)$ by rotating it into $r_L(1, 1)$ the element in the first row and first column of the upper triangular matrix $R_L(n)$. It can be constructed as

$$\hat{Q}_{(1)}(n) = \begin{bmatrix} \cos\theta_1(n) & \mathbf{0}^{\mathrm{T}} & \sin\theta_1(n) \\ \mathbf{0} & I & \mathbf{0} \\ -\sin\theta_1(n) & \mathbf{0}^{\mathrm{T}} & \cos\theta_1(n) \end{bmatrix} \quad (27)$$

where

$$\cos\theta_1(n) = \sqrt{\lambda}r_L(1, 1)/\sqrt{\lambda r_L^2(1, 1) + x_1^2(n)} \quad (28)$$

and

$$\sin\theta_1(n) = x_1(n)/\sqrt{\lambda r_L^2(1, 1) + x_1^2(n)}. \quad (29)$$

The other rotations can be calculated in a similar fashion. $\underline{\boldsymbol{U}}(n)$ and $\underline{\boldsymbol{V}}(n)$ can be recursively updated, using (24), as

$$\begin{aligned}\hat{Q}_L(n) \begin{bmatrix} \sqrt{\lambda}\underline{\boldsymbol{U}}(n-1) \\ \sqrt{\lambda}\underline{\boldsymbol{V}}(n-1) \\ d(n) \end{bmatrix} &= \begin{bmatrix} \underline{\boldsymbol{U}}(n) \\ \sqrt{\lambda}\underline{\boldsymbol{V}}(n-1) \\ \alpha(n) \end{bmatrix} \\ &= \begin{bmatrix} \underline{\boldsymbol{U}}(n) \\ \underline{\boldsymbol{V}}(n) \end{bmatrix}. \end{aligned} \quad (30)$$

The element $\alpha(n)$ in the equation above can be shown to be related to the estimation error. The current estimation error $e(n)$ is the last element of the error vector $\underline{\boldsymbol{e}}(n)$. It can be calculated as the inner product of $\underline{\boldsymbol{e}}(n)$ and the pinning vector $\underline{\boldsymbol{\sigma}}(n) = [0, \cdots, 0, 1]^{\mathrm{T}}$.

$$e(n) = \underline{\boldsymbol{\sigma}}^{\mathrm{T}}(n)B^{1/2}(n)\underline{\boldsymbol{e}}(n). \quad (31)$$

It is relatively straightforward to show [6], [21] that

$$e(n) = \gamma_L(n)\alpha(n) \quad (32)$$

where

$$\gamma_L(n) = \prod_{i=1}^{L} \cos(\theta_i(n)). \quad (33)$$

In many applications (for example, channel equalization, noise cancellation, etc.) it is important to evaluate the estimate of the desired response signal or the estimation error itself rather than the coefficients of the direct form representation of the nonlinear model. We now proceed to develop a technique for efficient evaluation of the estimation error signals for use

in such applications. The derivations of the fast QR-RLS algorithms, like most derivations of other fast RLS algorithms, involve solving the forward and backward prediction problems. Forward prediction involves estimating the matrix

$$\underline{\boldsymbol{d}}_f(n) = [\underline{\boldsymbol{x}}_1(n), \underline{\boldsymbol{x}}_2(n), \cdots, \underline{\boldsymbol{x}}_K(n)] \tag{34}$$

using the data matrix $X_L(n-1)$. The backward prediction problem involves estimating the matrix

$$\underline{\boldsymbol{d}}_b(n) = [\underline{\boldsymbol{x}}_1(n-N_1), \underline{\boldsymbol{x}}_2(n-N_2), \cdots, \underline{\boldsymbol{x}}_K(n-N_K)] \tag{35}$$

using the data matrix $X_L(n)$. Clearly, the forward and backward prediction problems are closely related since $\underline{\boldsymbol{d}}_f(n)$, $X_L(n-1)$, $\underline{\boldsymbol{d}}_b(n)$, and $X_L(n)$ are all contained in the augmented data matrix $X_{L+K}(n)$ which is defined by appending K additional columns to $X_L(n-1)$ or $X_L(n)$ as

$$\begin{aligned} X_{L+K}(n) &= [X_L(n) \vdots \underline{\boldsymbol{x}}_1(n-N_1), \cdots, \underline{\boldsymbol{x}}_K(n-N_K)] \\ &= [X_L(n-1) \vdots \underline{\boldsymbol{x}}_1(n), \cdots, \underline{\boldsymbol{x}}_K(n)]P \end{aligned} \tag{36}$$

where the P is a permutation matrix used to move the columns $\boldsymbol{x}_i(n)$ to their appropriate positions.

In QRD-based fast RLS algorithms, two equations are obtained for computing the orthogonal matrix $Q_{L+K}(n)$ that triangularizes $X_{L+K}(n)$, using the partitions given in (36). One of the equations for $Q_{L+K}(n)$ is obtained by solving the forward prediction problem using the first partition in (36). It depends explicitly on the orthogonal matrix $Q_L(n-1)$ from time n. The other equation is derived by solving the backward prediction problem using the second partition in equation (36). This equation depends explicitly on the matrix $Q_L(n)$ which is the desired orthogonal matrix at time n. These two equations for computing $Q_{L+K}(n)$ are used to solve for $Q_L(n+1)$, which is required at the next time step.

2.1 Backward Prediction

In the Lth order backward prediction problem, at time $n-1$, the desired data matrix $\underline{\boldsymbol{d}}_b(n-1)$ is estimated using the data matrix $X_L(n-1)$. The corresponding error matrix is given by

$$\underline{\boldsymbol{e}}_b(n-1) = \underline{\boldsymbol{d}}_b(n-1) - X_L(n-1)\underline{\boldsymbol{W}}_b(n-1). \tag{37}$$

The $L \times K$ element backward prediction coefficient matrix $\underline{\boldsymbol{W}}_b(n-1)$ is chosen to minimize the exponentially weighted least squares cost function

$$\begin{aligned} &\zeta(n-1) \\ &= \mathrm{Tr}\{[B^{1/2}(n-1)\underline{\boldsymbol{e}}_b(n-1)]^{\mathrm{T}}[B^{1/2}(n-1)\underline{\boldsymbol{e}}_b(n-1)]\} \end{aligned} \tag{38}$$

where $\mathrm{Tr}\{(\cdot)\}$ denotes the trace of the matrix $(\cdot)$. Let

$$Q_L(n-1)B^{1/2}(n-1)\boldsymbol{d}_b(n-1) = \begin{bmatrix} \underline{\boldsymbol{U}}_b(n-1) \\ \underline{\boldsymbol{V}}_b(n-1) \end{bmatrix} \tag{39}$$

where $\underline{\boldsymbol{U}}_b(n-1)$ and $\underline{\boldsymbol{V}}_b(n-1)$ by definition have L and $n-L$ rows, respectively.

Now consider triangularizing the augmented matrix $X_{L+K}(n)$ partitioned as

$$X_{L+K}(n-1) = [X_L(n-1) \vdots \underline{\boldsymbol{d}}_b(n-1)]. \tag{40}$$

Using equations (21) and (39), we obtain

$$\begin{aligned} &Q_L(n-1)B^{1/2}(n-1)X_{L+K}(n-1) \\ &\qquad = \begin{bmatrix} R_L(n-1) & \underline{\boldsymbol{U}}_b(n-1) \\ \mathbf{0} & \underline{\boldsymbol{V}}_b(n-1) \end{bmatrix}. \end{aligned} \tag{41}$$

The triangularization can be completed by operating on both sides of (41) in such a way that $\underline{\boldsymbol{V}}_b(n-1)$ is transformed into a $K \times K$ element upper triangular matrix $\underline{\boldsymbol{\epsilon}}_b(n-1)$. Let $Q_b(n-1)$ be an orthogonal matrix that effects this transformation, i.e.,

$$\begin{aligned} &Q_b(n-1)\begin{bmatrix} R_L(n-1) & \underline{\boldsymbol{U}}_b(n-1) \\ \mathbf{0} & \underline{\boldsymbol{V}}_b(n-1) \end{bmatrix} \\ &\qquad = \begin{bmatrix} R_L(n-1) & \underline{\boldsymbol{U}}_b(n-1) \\ \mathbf{0} & \underline{\boldsymbol{\epsilon}}_b(n-1) \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \\ &\qquad = \begin{bmatrix} R_{L+K}(n-1) \\ \mathbf{0} \end{bmatrix}. \end{aligned} \tag{42}$$

Clearly, $Q_b(n-1)$ has no effect on the matrices $R_L(n-1)$ and $\underline{\boldsymbol{U}}_b(n-1)$. In order to recursively triangularize the augmented data matrix, consider $X_{L+K}(n)$ given by

$$X_{L+K}(n) = \begin{bmatrix} X_L(n-1) & \boldsymbol{d}_b(n-1) \\ \underline{\boldsymbol{z}}_1 & \underline{\boldsymbol{z}}_2 \end{bmatrix} \tag{43}$$

whose top n rows form $X_{L+K}(n-1)$ and the last row is the new data that appears at time n. That is,

$$\underline{\boldsymbol{z}}_1 = [x_1(n), \cdots, x_K(n-N_K+1)], \tag{44}$$

$$\underline{\boldsymbol{z}}_2 = [x_1(n-N_1), \cdots, x_K(n-N_K)]. \tag{45}$$

Let $\hat{Q}_L(n)$ be the orthogonal matrix that rotates $\underline{\boldsymbol{z}}_1$ into $\sqrt{\lambda}R_L(n-1)$ so that

$$\begin{aligned} &\hat{Q}_L(n)\begin{bmatrix} \sqrt{\lambda}R_L(n-1) & \sqrt{\lambda}\underline{\boldsymbol{U}}_b(n-1) \\ \mathbf{0} & \sqrt{\lambda}\underline{\boldsymbol{\epsilon}}_b(n-1) \\ \mathbf{0} & \mathbf{0} \\ \underline{\boldsymbol{z}}_1 & \underline{\boldsymbol{z}}_2 \end{bmatrix} \\ &\qquad = \begin{bmatrix} R_L(n) & \underline{\boldsymbol{U}}_b(n) \\ \mathbf{0} & \sqrt{\lambda}\underline{\boldsymbol{\epsilon}}_b(n-1) \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0}^{\mathrm{T}} & \underline{\boldsymbol{\alpha}}_b(n-1) \end{bmatrix}. \end{aligned} \tag{46}$$

Note that this operation has transformed $\sqrt{\lambda}\underline{\boldsymbol{U}}_b(n-1)$ to $\underline{\boldsymbol{U}}_b(n)$ and $\underline{\boldsymbol{z}}_2$ to $\underline{\boldsymbol{\alpha}}_b(n-1)$. Now, define $\hat{Q}_b(n)$ as the rotation matrix that annihilates $\underline{\boldsymbol{\alpha}}_b(n-1)$ by rotating it into the upper triangular matrix $\sqrt{\lambda}\underline{\boldsymbol{\epsilon}}_b(n-1)$, which is transformed to $\underline{\boldsymbol{\epsilon}}_b(n)$, and thus completing the triangularization.

$$\hat{Q}_b(n)\begin{bmatrix} R_L(n) & \underline{\boldsymbol{U}}_b(n) \\ \mathbf{0} & \sqrt{\lambda}\underline{\boldsymbol{\epsilon}}_b(n-1) \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0}^{\mathrm{T}} & \underline{\boldsymbol{\alpha}}_b(n-1) \end{bmatrix} = \begin{bmatrix} R_{L+K}(n) \\ \mathbf{0} \end{bmatrix}. \tag{47}$$

The above discussion implies that $Q_{L+K}(n)$ that triangularizes the augmented and weighted data matrix $B^{1/2}(n)X_{L+K}(n)$ can be obtained recursively as

$$\begin{aligned} Q_{L+K}(n) &= \hat{Q}_b(n)\hat{Q}_L(n) \\ &\quad \cdot \begin{bmatrix} Q_b(n-1) & \mathbf{0} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix}\begin{bmatrix} Q_L(n-1) & \mathbf{0} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix}. \end{aligned} \tag{48}$$

In the next section we derive another update equation for $Q_{L+K}(n)$ which depends on $Q_L(n)$. Knowing $Q_L(n-1)$, we can then solve for $Q_L(n)$ by first computing $Q_{L+K}(n)$ and then using the new relationship to find $Q_L(n)$.

2.2 Forward Prediction

Similar to (37), the forward prediction error matrix $\boldsymbol{e}_f(n-1)$ is given by

$$\underline{\boldsymbol{e}}_f(n-1) = \underline{\boldsymbol{d}}_f(n-1) - X_L(n-2)\underline{\boldsymbol{W}}_f(n-1) \tag{49}$$

where the $L \times K$ element forward prediction coefficient matrix $\underline{\boldsymbol{W}}_f(n-1)$ is chosen to minimize the LS cost function

$$\zeta_f(n-1) = \mathrm{Tr}\{[B^{1/2}(n-2)\underline{\boldsymbol{e}}_f(n-1)]^{\mathrm{T}} \cdot [B^{1/2}(n-2)\underline{\boldsymbol{e}}_f(n-1)]\}. \tag{50}$$

Similar to (39), let

$$Q_L(n-2)B^{1/2}(n-2)\underline{\boldsymbol{d}}_f(n-1) = \begin{bmatrix} \underline{\boldsymbol{U}}_f(n-1) \\ \underline{\boldsymbol{V}}_f(n-1) \end{bmatrix} \tag{51}$$

where the right-hand side is partitioned as before. Again, consider triangularizing the data matrix $X_{L+K}(n-1)$ using the solution to the forward prediction problem. Since the solution to the forward prediction problem is to be used to triangularize the augmented matrix, it is partitioned as

$$X_{L+K}(n-1) = [X_L(n-2) \vdots \underline{\boldsymbol{x}}_1(n-1), \cdots, \underline{\boldsymbol{x}}_K(n-1)]P \tag{52}$$

where P, as defined earlier, is a permutation matrix that is required to undo the permutations of the columns of $X_{L+K}(n-1)$. Partial triangularization is achieved by operating with $Q_L(n-2)$ which triangularizes $X_L(n-2)$. Define $Q_f^v(n-1)$ to be the orthogonal matrix that annihilates $\sqrt{\lambda}[x_1(1), \cdots, x_K(1)]$ and the matrix $\underline{\boldsymbol{V}}_f(n-1)$ by rotating them into a $K \times K$ element upper triangular matrix $\boldsymbol{\epsilon}_f(n-1)$. It is straightforward to see that operating with $Q_f^v(n-1)$ and $Q_L(n-2)$ on the appropriate data matrix will result in the following:

$$Q_f^v(n-1)\begin{bmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ \mathbf{0} & Q_L(n-2) \end{bmatrix} B^{1/2}(n-1)X_{L+K}(n-1)P = \begin{bmatrix} \mathbf{0}^{\mathrm{T}} & \mathbf{0}^{\mathrm{T}} \\ R_L(n-2) & \underline{\boldsymbol{U}}_f(n-1) \\ \mathbf{0} & \underline{\boldsymbol{\epsilon}}_f(n-1) \\ \mathbf{0} & \mathbf{0} \end{bmatrix} P. \tag{53}$$

The matrix on the right-hand side of the preceding equation is not triangular since P permutes the columns of the triangular matrix on which it operates. In order to complete the triangularization we need to operate on the above equation with another set of rotations (we use $Q_u^f(n-1)$ to denote the orthogonal matrix that performs these rotations). Note that $\boldsymbol{U}_f(n-1)$ is an $L \times K$ element matrix and $\underline{\boldsymbol{\epsilon}}_f(n-1)$ is a $K \times K$ element upper triangular matrix. Hence, the ith column of the submatrix

$$\begin{bmatrix} \mathbf{0} \\ \underline{\boldsymbol{U}}_f(n-1) \\ \underline{\boldsymbol{\epsilon}}_f(n-1) \end{bmatrix} \tag{54}$$

has $L+i$ nonzero elements. Now, the permutation matrix P moves the ith column of the above submatrix to the column $i + \sum_{j=0}^{i-1}(j)(N_j - N_{j+1})$ (assume that $N_0 = 0$) of the triangular matrix $R_L(n-1)$. This implies that $Q_f^u(n-1)$ has to annihilate $L - \sum_{j=0}^{i-1}(j)(N_j - N_{j+1})$ elements from the ith column of (54). The rotations that constitute $Q_j^u(n-1)$ can be computed as they are based on the elements of the above submatrix and not on the triangular matrix $R_L(n-2)$. Hence the triangular matrix need not be saved.

Now consider recursively triangularizing the augmented data matrix $X_{L+K}(n)$. Let us partition $X_{L+K}(n)$ as

$$X_{L+K}(n) = \begin{bmatrix} X_L(n-2) & [\boldsymbol{x}_1(n-1), \cdots, \boldsymbol{x}_K(n-1)] \\ \underline{\boldsymbol{s}}_1 & \underline{\boldsymbol{s}}_2 \end{bmatrix} P \tag{55}$$

where

$$\underline{\boldsymbol{s}}_1 = [x_1(n-1), \cdots, x_K(n-NK)] \tag{56}$$

and

$$\underline{\boldsymbol{s}}_2 = [x_1(n), \cdots, x_K(n)] \tag{57}$$

represent the new data that appears in $X_L + K(n)$ at time n. Operating with $Q_L(n-2)$ and $Q_f^v(n-1)$ and using (53), we get

$$\begin{bmatrix} Q_f^v(n-1) & \mathbf{0} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix} \begin{bmatrix} 1 & \mathbf{0}^{\mathrm{T}} & 0 \\ \mathbf{0} & Q_L(n-2) & \mathbf{0} \\ 0 & \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix} B^{1/2}(n)X_{L+K}(n) = \begin{bmatrix} \mathbf{0}^{\mathrm{T}} & \mathbf{0}^{\mathrm{T}} \\ \sqrt{\lambda}R_L(n-2) & \sqrt{\lambda}\underline{\boldsymbol{U}}_f(n-1) \\ \mathbf{0} & \sqrt{\lambda}\underline{\boldsymbol{\epsilon}}_f(n) \\ \mathbf{0} & \mathbf{0} \\ \underline{\boldsymbol{s}}_1 & \underline{\boldsymbol{s}}_2 \end{bmatrix} P. \tag{58}$$

From (26) it is clear that the vector $\boldsymbol{s}_1$ can be annihilated by $\hat{Q}_L(n-1)$. Hence,

$$\begin{bmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ \mathbf{0} & \hat{Q}_L(n-1) \end{bmatrix} \begin{bmatrix} \mathbf{0}^{\mathrm{T}} & \mathbf{0}^{\mathrm{T}} \\ \sqrt{\lambda}R_L(n-2) & \sqrt{\lambda}\underline{\boldsymbol{U}}_f(n-1) \\ \mathbf{0} & \sqrt{\lambda}\underline{\boldsymbol{epsilon}}_f(n-1) \\ \mathbf{0} & \mathbf{0} \\ \underline{\boldsymbol{s}}_1 & \underline{\boldsymbol{s}}_2 \end{bmatrix} P = \begin{bmatrix} \mathbf{0}^{\mathrm{T}} & \mathbf{0}^{\mathrm{T}} \\ R_L(n-1) & \underline{\boldsymbol{U}}_f(n) \\ \mathbf{0} & \sqrt{\lambda}\underline{\boldsymbol{\epsilon}}_f(n-1) \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0}^{\mathrm{T}} & \underline{\boldsymbol{\alpha}}_f(n) \end{bmatrix} P \tag{59}$$

where $\underline{\boldsymbol{\alpha}}_f(n)$ is a $1 \times K$ element row vector. Now, define $\hat{Q}_f^v(n)$ as the orthogonal matrix that annihilates $\underline{\boldsymbol{\alpha}}_f(n)$ by rotating it into the $K \times K$ element upper triangular matrix $\sqrt{\lambda}\underline{\boldsymbol{\epsilon}}_f(n-1)$. See (60) at bottom of the next page. Again, the matrix on the right-hand side is not triangular because of the permutation matrix P. The triangularization is completed with $Q_f^u(n)$. $Q_f^u(n)$ is computed as explained earlier.

$$\hat{Q}_f^u(n)\begin{bmatrix} \mathbf{0}^{\mathrm{T}} & \mathbf{0}^{\mathrm{T}} \\ R_L(n-1) & \underline{\boldsymbol{U}}_f(n) \\ \mathbf{0} & \underline{\boldsymbol{\epsilon}}_f(n) \\ \mathbf{0} & \mathbf{0} \end{bmatrix} P = \begin{bmatrix} \mathbf{0}^{\mathrm{T}} \\ R_{L+K}(n) \\ \mathbf{0} \end{bmatrix}. \tag{61}$$

Finally, in order to make the correspondence of the resultant triangular matrix exact with that obtained in (47), we need to shift the upper triangular matrix to the top L rows of the matrix on the right-hand side of (61). This can be achieved by operating on (61) with an appropriate shift matrix J.

The above sequence of rotations (58)–(61) lead to another equation for computing the orthogonal matrix $Q_{L+K}(n)$.

$$Q_{L+K}(n) = JQ_f^u(n)\hat{Q}_f^v(n)\begin{bmatrix}1 & \mathbf{0}^{\mathrm{T}} \\ \mathbf{0} & \hat{Q}_L(n-1)\end{bmatrix} \cdot \begin{bmatrix}Q_f^v(n-1) & \mathbf{0} \\ \underline{\mathbf{0}}^{\mathrm{T}} & 1\end{bmatrix}\begin{bmatrix}1 & \mathbf{0}^{\mathrm{T}} & 0 \\ \underline{\mathbf{0}} & Q_L(n-2) & \underline{\mathbf{0}} \\ 0 & \mathbf{0}^{\mathrm{T}} & 1\end{bmatrix}. \quad (62)$$

As is shown in the next section, (48) and (62) are essential in deriving the fast algorithm; however, not all the orthogonal rotations in (48) and (62) need to be computed. $\hat{Q}_f^v(n)$ and $Q_f^u(n)$ are the only orthogonal rotations that have to be calculated in order to update $\hat{Q}_L(n-1)$ to $\hat{Q}_L(n)$. As mentioned earlier, (62) is used to calculate $\underline{\boldsymbol{\alpha}}_L(n)$ and $\gamma_L(n)$ and then $\hat{Q}_L(n)$ is computed.

2.3 Algorithm

Equations (48) and (62) provide us with a way of calculating the a posteriori estimation error in a recursive manner. Given $\hat{Q}_L(n-1)$, calculate $Q_{L+K}(n)$ as in (62). Then solve for $\hat{Q}_L(n)$ using (48). Having obtained $\hat{Q}_L(n)$, we can recursively compute $e(n)$ from (30)–(33). Note that the key is to obtain the L rotations contained in $\hat{Q}_L(n)$ as efficiently as possible. In the process, if we can avoid calculating all or at least some of the intermediate rotations in (48) and (62), we would have developed a much more efficient algorithm. In order to do this, let us look at the last column of $Q_{L+K}(n)$ as given by (48) and (62). Obviously, the last column can be obtained by postmultiplying $Q_{L+K}(n)$ with the pinning vector $\underline{\boldsymbol{\sigma}}(n)$. Operating on (48) gives

$$Q_{L+K}(n)\underline{\boldsymbol{\sigma}}(n) = \hat{Q}_b(n)\hat{Q}_L(n)\underline{\boldsymbol{\sigma}}(n). \quad (63)$$

It is easy to show by direct calculation that

$$\hat{Q}_L(n)\underline{\boldsymbol{\sigma}}(n) = \begin{bmatrix}\underline{\boldsymbol{\alpha}}_L(n) \\ \mathbf{0} \\ \gamma_L(n)\end{bmatrix} \quad (64)$$

where $\underline{\boldsymbol{\alpha}}_L(n)$ is a vector of L elements whose ith element is given by

$$a_{i,L}(n) = \begin{cases}\sin\theta_i(n), & i = 1 \\ \sin\theta_i(n)\prod_{j=1}^{i-1}\cos\theta_j(n), & \text{otherwise}\end{cases} \quad (65)$$

and

$$\gamma_L(n) = \prod_{i=1}^{L}\cos\theta_i(n). \quad (66)$$

In the above, the angles $\theta_j(n)$, $j = 1, \cdots, L$, represent the L rotations that constitute the orthogonal matrix $\hat{Q}_L(n)$. Note that one can solve for the rotations if $\underline{\boldsymbol{\alpha}}_L(n)$ is known. Notice also that, since orthogonal rotations are length preserving, and since $\underline{\boldsymbol{\sigma}}(n)$ has unit length, one can compute $\gamma_L(n)$ as

$$\gamma_L(n) = \sqrt{1 - \|\underline{\boldsymbol{\alpha}}_L(n)\|^2}. \quad (67)$$

Now, $\hat{Q}_b(n)$ rotates the last row of the matrix it operates on into the K rows immediately below the first L rows. In particular, operating with $\hat{Q}_b(n)$ on $\hat{Q}_L(n)\underline{\boldsymbol{\sigma}}(n)$, as in (63), does not affect $\underline{\boldsymbol{\alpha}}_L(n)$. Let

$$Q_{L+K}(n)\boldsymbol{\sigma}(n) = \begin{bmatrix}\underline{\boldsymbol{\alpha}}_{L+K}(n) \\ \mathbf{0} \\ \gamma_{L+K}(n)\end{bmatrix} \quad (68)$$

where $\underline{\boldsymbol{\alpha}}_{L+K}(n)$ and $\gamma_{L+K}(n)$ are defined in a similar manner to (65) and (66). It is clear from the above discussion that the top L components of $\underline{\boldsymbol{\alpha}}_{L+K}(n)$ are identical to $\underline{\boldsymbol{\alpha}}_L(n)$.

Using (62), and again operating on $\underline{\boldsymbol{\sigma}}(n)$, leads to

$$Q_{L+K}(n)\underline{\boldsymbol{\sigma}}(n) = JQ_f^u(n)\hat{Q}_f^v(n)\begin{bmatrix}0 \\ \underline{\boldsymbol{\alpha}}_L(n-1) \\ \mathbf{0} \\ \gamma_L(n-1)\end{bmatrix}. \quad (69)$$

Now, $\hat{Q}_f^v(n)$ rotates $\gamma_L(n-1)$ into the K zeros below the vector $\underline{\boldsymbol{\alpha}}_L(n-1)$. Let $\underline{\boldsymbol{h}}(n-1)$ denote the K element vector so generated. Also, note that the rotations transform $\gamma_L(n-1)$ into $\gamma_{L+K}(n)$ (J and $Q_u^f(n)$ do not affect the last row).

$$\hat{Q}_f^v(n)\begin{bmatrix}0 \\ \underline{\boldsymbol{\alpha}}_L(n-1) \\ \mathbf{0} \\ \gamma_L(n-1)\end{bmatrix} = \begin{bmatrix}0 \\ \underline{\boldsymbol{\alpha}}_L(n-1) \\ \underline{\boldsymbol{h}}(n-1) \\ \mathbf{0} \\ \gamma_{L+K}(n)\end{bmatrix}. \quad (70)$$

Finally, $Q_f^u(n)$ rotates $\underline{\boldsymbol{h}}(n-1)$ into $\underline{\boldsymbol{\alpha}}_L(n-1)$ to generate $\underline{\boldsymbol{\alpha}}_{L+K}(n)$.

$$JQ_f^u(n)\begin{bmatrix}0 \\ \underline{\boldsymbol{\alpha}}_L(n-1) \\ \underline{\boldsymbol{h}}(n-1) \\ \mathbf{0} \\ \gamma_{L+K}(n)\end{bmatrix} = \begin{bmatrix}\underline{\boldsymbol{\alpha}}_{L+K}(n) \\ \mathbf{0} \\ \gamma_{L+K}(n)\end{bmatrix}. \quad (71)$$

$$\hat{Q}_f^v(n)\begin{bmatrix}\mathbf{0}^{\mathrm{T}} & \mathbf{0}^{\mathrm{T}} \\ R_L(n-1) & \underline{\boldsymbol{U}}_f(n) \\ \mathbf{0} & \sqrt{\lambda}\underline{\boldsymbol{\epsilon}}_f(n-1) \\ \mathbf{0} & \mathbf{0}^{\mathrm{T}} \\ \mathbf{0}^{\mathrm{T}} & \underline{\boldsymbol{\alpha}}_f(n)\end{bmatrix}P = \begin{bmatrix}\mathbf{0}^{\mathrm{T}} & \mathbf{0}^{\mathrm{T}} \\ R_L(n-1) & \underline{\boldsymbol{U}}_f(n) \\ \mathbf{0} & \underline{\boldsymbol{\epsilon}}_f(n+1) \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0}^{\mathrm{T}} & \mathbf{0}^{\mathrm{T}}\end{bmatrix}P. \quad (60)$$

TABLE I
BLOCK VERSION OF THE QR-RLS ADAPTIVE SECOND-ORDER VOLTERRA FILTER*

Given $d(n), x(n), \cdots, x(n-N+1)$, $\mathrm{U}_f(n-1)$, $\xi_f(n-1)$, $\mathrm{a}_L(n-1)$, $\gamma_L(n-1)$, and $\hat{Q}_L(n-1)$.

DO STEPS 1 - 9 for $n = 1$ onwards.

STEP #	OPERATION	EQUATION IN PAPER
1	$x_i(n) = \begin{cases} x(n) & i = 1 \\ x(n)x(n-i+2), & i = 2, \cdots, N+1 \end{cases}$	(10)
2	$\begin{bmatrix} \mathrm{U}_f(n) \\ \underline{\alpha}_f(n) \end{bmatrix} = \hat{Q}_L(n-1) \begin{bmatrix} \sqrt{\lambda}\mathrm{U}_f(n-1) \\ \boldsymbol{x}_1(n), \cdots, \boldsymbol{x}_K(n) \end{bmatrix}$	(59)
3	Find $\hat{Q}_f^{\mathrm{H}}(n)$ that annihilates $\underline{\alpha}_f(n)$ by rotating it into $\sqrt{\lambda}\xi_f(n-1)$. $\begin{bmatrix} \xi_f(n) \\ 0 \end{bmatrix} = \hat{Q}_f^{\mathrm{H}}(n) \begin{bmatrix} \sqrt{\lambda}\xi_f(n-1) \\ \underline{\alpha}_f(n) \end{bmatrix}$	(60)
	$\begin{bmatrix} \mathrm{b}(n-1) \\ \gamma_{L+K}(n) \end{bmatrix} = \hat{Q}_f^{\mathrm{H}}(n) \begin{bmatrix} 0 \\ \gamma_L(n-1) \end{bmatrix}$	(70)
4	Find $Q_f^{\mathrm{H}}(n)$ that annihilates $L - \sum_{j=0}^{i-1} j(N_j - N_{j+1})$ elements from the $i-th$ column of the matrix on which it operates. $\begin{bmatrix} * \\ 0 \end{bmatrix} = Q_f^{\mathrm{H}}(n) \begin{bmatrix} \mathrm{U}_f(n) \\ \xi_f(n) \end{bmatrix}$	(61)
5	The top L components of $\mathrm{a}_{L+K}(n)$ constitute $\mathrm{a}_L(n)$. $\mathrm{a}_{L+K}(n) = Q_f^{\mathrm{H}}(n) \begin{bmatrix} \mathrm{a}_L(n-1) \\ \mathrm{b}(n-1) \end{bmatrix}$	(71)
6	$\gamma_L(n) = +\sqrt{1 - \|\mathrm{a}_L(n)\|^2}$	(67)
7	Compute the rotations that define $Q_L(n)$. $\begin{bmatrix} \mathrm{a}_L(n) \\ \gamma_L(n) \end{bmatrix} = \hat{Q}_L(n)\underline{x}(n)$	(64)
8	$\begin{bmatrix} \mathrm{U}(n) \\ \alpha(n) \end{bmatrix} = \hat{Q}_L(n) \begin{bmatrix} \sqrt{\lambda}\mathrm{U}(n-1) \\ d(n) \end{bmatrix}$	(30)
9	$e(n) = \alpha(n)\gamma_L(n)$	(32)

*The sizes of the matrices that define the rotations are smaller than in the text and are also fixed in the table since the irrelevant columns of the data matrices have been omitted here.

We now have all the equations necessary for computing $\hat{Q}_L(n)$ more efficiently than direct calculation. The relevant steps in the right order are tabulated in Table I.

The algorithm presented in Table I requires $O(N^4)$ arithmetic operations per time step. The main computational burden comes at Step 4 which involves slightly less than $L(N+1)^2$ rotations. This complexity is an order of magnitude larger than that of fast RLS adaptive Volterra filters presented in [8], [20]. Consequently, it is desirable to reduce the complexity of our algorithm further. Our approach for achieving this objective is to develop a variation of the algorithm that processes different channels "sequentially" at each time instant. The resulting algorithm is computationally more efficient than the algorithm of Table I because it does not require any matrix processing. The approach is similar to that presented in [18] for multichannel filtering. However, our formulation of the nonlinear filtering problem requires a different number of coefficients for different channels and is therefore more general than all previous work. Another advantage of the sequential processing algorithm is that it corresponds exactly to processing in K single-channel algorithms one after the other. Hence knowledge of the single-channel algorithm can be carried over to the multichannel algorithm. Also, sequential processing leads to regularity in implementation which results in a modular architecture [9], [18]. The derivation of the algorithm is very similar to the derivation of the block algorithm. Hence, the details of the derivation will not be presented. However, some new notation, appropriate for sequential processing, needs to be introduced.

Define the data matrix $X_L(n)$ as

$$X_L(n) = [\underline{\boldsymbol{x}}_1(n) \cdots \underline{\boldsymbol{x}}_1(n-(N_1-N_2)+1), \underline{\boldsymbol{x}}_2(n) \cdots \underline{\boldsymbol{x}}_K(n-N_K+1) \cdots \underline{\boldsymbol{x}}_1(n-N_1+1)]. \quad (72)$$

From the definition it is clear that in going from time $n-1$ to n the column vectors $\underline{\boldsymbol{x}}_i(n)$, $i = 1, \cdots, K$, are appended to $X_L(n-1)$ and the column vectors $\underline{\boldsymbol{x}}_i(n-N_i)$, $i = 1, \cdots, K$, are removed from the data matrix. In the block algorithm presented in the previous section the K column vectors $\underline{\boldsymbol{x}}_i(n)$ were all appended simultaneously and the K column vectors $\underline{\boldsymbol{x}}_i(n+N_i)$ were all removed at the same time. The sequential processing algorithm generates the data matrix $X_L(n)$ from $X_L(n-1)$ by substituting one column at a time. In order to illustrate this, let us define the following data matrices:

$$X_{L-1}(n-1) = [\underline{\boldsymbol{x}}_1(n-1) \cdots \underline{\boldsymbol{x}}_1(n-(N_1-N_2)), \underline{\boldsymbol{x}}_2(n-1) \cdots \underline{\boldsymbol{x}}_K(n-N_K+1) \cdots \underline{\boldsymbol{x}}_1(n-N_1+1)] \quad (73)$$

$$X_{i,L,n} = [X_{L-1}(n-1) \vdots \underline{\boldsymbol{x}}_K(n-N_K), \cdots, \underline{\boldsymbol{x}}_{i+1}(n-N_{i+1}) \vdots \underline{\boldsymbol{x}}_1(n), \cdots, \underline{\boldsymbol{x}}_i(n)]S_{L,i} \quad (74)$$

and

$$X_{i,L+1,n} = [X_{L-1}(n-1) \vdots \boldsymbol{x}_K(n-N_K), \cdots, \boldsymbol{x}_i(n-N_i) \vdots \boldsymbol{x}_1(n), \cdots, \boldsymbol{x}_i(n)]S_{L+1,i} \quad (75)$$

where $S_{L,i}$ and $S_{L+1,i}$, $i = 1, \cdots, N+1$, are permutation matrices that move the column vectors $\underline{\boldsymbol{x}}_i(n)$ to their appropriate places. Note that $X_{i,L,n}$ contains i column vectors $\underline{\boldsymbol{x}}_j(n)$, $j = 1, \cdots, i$, that belong to $X_L(n)$ but not to $X_L(n-1)$. It also contains $K-i$ column vectors $\underline{\boldsymbol{x}}_K(n-N_K)$, $\underline{\boldsymbol{x}}_{K-1}(n-N_{K-1}), \cdots, \underline{\boldsymbol{x}}_{i+1}(n-N_{i+1})$, that belong to $X_L(n-1)$ but not to $X_L(n)$. We can obtain $X_{i+1,L,n}$ from $X_{i,L,n}$ by replacing $\underline{\boldsymbol{x}}_{i+1}(n-N_{i+1})$ in $X_{i,L,n}$ with $\underline{\boldsymbol{x}}_{i+1}(n)$. Starting with $X_{0,L,n} = X_L(n-1)$, it is straightforward to see that iterating K times as above will give us $X_{K,L,n} = X_L(n)$.

An efficient algorithm for adaptive Volterra filtering can be obtained by triangularizing $X_{i,L,n}$ in a sequential manner, beginning with $X_{0,L,n}$ and ending with $X_{K,L,n}$ at time n. Given that $X_{i-1,L,n}$ has been triangularized, we can triangularize $X_{i,L,n}$ by first triangularizing the augmented data matrix given by

$$X_{i,L+1,n} = [X_{i,L,n} \vdots \underline{\boldsymbol{x}}_i(n-N_i)] = [X_{i-1,L,n} \vdots \underline{\boldsymbol{x}}_i(n)]P_i. \quad (76)$$

In the above, P_i is an appropriate premutation matrix that preserves the equality of the matrices on the right-hand side of (76).

The problem of updating the rotations from iteration $i-1$ to iteration i at time n is exactly like the procedure for time-updating the rotations for a single-channel problem. We can define the forward and backward prediction problems as those of estimating $x_i(n)$ and $x_i(n-N)$ with $X_{i-1,L,n}$ and $X_{i,L,n}$, respectively, and the two problems are related

TABLE II
SEQUENTIAL VERSION OF THE QR-RLS ADAPTIVE SECOND-ORDER VOLTERRA FILTER*

Given $d(n)$, $x(n), \cdots, x(n-N+1)$, $\mathrm{U}_i^f(n-1)$, $\xi_{i,L}^f(n-1)$, $\mathbf{a}_{i-1,L}(n)$, $\gamma_{i-1,L}(n)$, and $\hat{Q}_{i-1,L}(n)$.

DO STEPS 1 – 9 for $n = 1$, *onwards*.

STEP #	OPERATION	EQUATION IN PAPER
1	$x_i(n) = \begin{cases} x(n) & i = 1 \\ x(n)x(n-i+2), & i = 2, \cdots, N+1 \end{cases}$	(10)

DO STEPS 2 – 7 for $i = 1, \cdots, N+1$.

2	$\begin{bmatrix} \mathrm{U}_i^f(n) \\ \alpha_{i,L}^f(n) \end{bmatrix} = Q_{i-1,L}(n) \begin{bmatrix} \sqrt{\lambda}\mathrm{U}_i^f(n-1) \\ x_i(n) \end{bmatrix}$	(59)
3	Find $\hat{Q}_f^v(n)$ that annihilates $\underline{\alpha}_f(n)$ by rotating it into $\sqrt{\lambda}\underline{\epsilon}_f(n-1)$.	
	$\begin{bmatrix} \epsilon_{i,L}^f(n) \\ 0^T \end{bmatrix} = \hat{Q}_{i,L}^v(n) \begin{bmatrix} \sqrt{\lambda}\epsilon_{i,L}^f(n-1) \\ \alpha_{i,L}^f(n) \end{bmatrix}$	(60)
	$\begin{bmatrix} h_i(n-1) \\ \gamma_{i,L+1}(n) \end{bmatrix} = \hat{Q}_{i,L}^v(n) \begin{bmatrix} 0 \\ \gamma_{i-1,L}(n) \end{bmatrix}$	(70)
4	Find $Q_f^v(n)$ that annihilates $L - \sum_{j=0}^{i-1} j(N_j - N_{j+1})$ elements from the $i-th$ column of the matrix on which it operates.	
	$\begin{bmatrix} * \\ 0 \end{bmatrix} = Q_{i,L}^v(n) \begin{bmatrix} \mathrm{U}_{i,L}^f \\ \epsilon_{i,L}^f(n) \end{bmatrix}$, * denotes 'don't care'	(61)
5	The top L components of $\mathbf{a}_{L+K}(n)$ constitute $\mathbf{a}_L(n)$.	
	$\mathbf{a}_{i,L+1}(n) = Q_{i,L}^v(n) \begin{bmatrix} \mathbf{a}_{i-1,L}(n) \\ h_i(n-1) \end{bmatrix}$	(71)
6	$\gamma_{i,L}(n) = \sqrt{1 - \|\mathbf{a}_{i,L}(n)\|^2}$	(67)
7	Compute the rotations that define $\hat{Q}_{i,L}(n)$.	
	$\begin{bmatrix} \mathbf{a}_{i,L}(n) \\ \gamma_{i,L}(n) \end{bmatrix} = \hat{Q}_{i,L}(n)\underline{e}$	(64)
8	$\begin{bmatrix} \mathrm{U}(n) \\ \alpha(n) \end{bmatrix} = \hat{Q}_{K,L}(n) \begin{bmatrix} \sqrt{\lambda}\mathrm{U}(n-1) \\ d(n) \end{bmatrix}$	(30)
9	$e(n) = \gamma_{K,L}(n)\alpha(n)$	(32)

*The sizes of the matrices that define the rotations are smaller than in the text and are also fixed in the table since the irrelevant columns of the data matrices have been omitted here.

through the augmented data matrix $X_{i,L+1,n}$. As for the block processing algorithm, we can develop two different ways of triangularizing the augmented data matrix using the two partitions of $X_{i,L+1,n}$ in (76) and solve for the required rotations for the ith iteration from the two equations that result from these operations. The steps are identical to the derivation of the block processing algorithm and are omitted here. The algorithm is given in Table II. A subscript i has been added to every rotation matrix as well as several other variables to indicate explicitly that the operations are part of the ith iteration at time n. The implications of these variables are exactly the same as the corresponding ones, without the additional subscripts, in Table I. The computational complexity of each step of the algorithm is tabulated in Table III. The biggest savings over the block processing algorithm is in Step 4. There is an order of magnitude difference in the computational complexity associated with this step between the two algorithms. As a result of this, the overall computational complexity of the sequential algorithm is $O(N^3)$ per time step.

III. EXPERIMENTAL RRESULTS

In order to evaluate the performance of the algorithm, several experiments were performed. In these experiments the adaptive filters were used in the system identification mode.

TABLE III
OPERATIONS COUNT FOR THE QR-RLS ADAPTIVE SECOND-ORDER VOLTERRA FILTER WITH SEQUENTIAL PROCESSING

Step #	# of Multipications	# of Square Roots
(1)	N	
(2)	$\frac{5N^3}{2} + 10N^2 + \frac{15N}{2}$	
(3 – 1)	$5(N+1)$	$N+1$
(3 – 2)	$N+1$	
(4)	$2N^3 + 6N^2 - 4$	$\frac{N^3}{2} + \frac{3N^2}{2} - 1$
(5)	$2N^3 + 6N^2 - 4$	
(6)	$\frac{N^3}{2} + 2N^2 + \frac{3N}{2}$	$N+1$
(7)	$2N^3 + 8N^2 + 6N$	$\frac{N^2}{2} + 2N^2 + \frac{3N}{2}$
(8)	$\frac{5N^2}{2} + \frac{15N}{2}$	
(9)	1	
Total	$9N^3 - \frac{69N^2}{2} + \frac{57N}{2} - 1$	$N^3 + \frac{7N^2}{2} + \frac{7N}{2}$

TABLE IV
LINEAR AND QUADRATIC COEFFICIENTS OF THE UNKNOWN SYSTEM USED IN THE SYSTEM IDENTIFICATION EXPERIMENTS

Linear Coefficients a_i.

i	0	1	2	3	4	5	6	7	8	9
a_i	-0.052	0.723	0.435	-0.196	-0.143	0.812	0.354	0.077	-1.379	2.251

Quadratic coefficients b_{ij}.

i/j	0	1	2	3	4	5	6	7	8	9
0	1.020	-1.812	-1.138	-0.592	-0.144	-0.966	-1.454	1.820	-4.022	5.562
1		1.389	-2.608	-1.486	-1.382	-2.308	4.256	0.626	-0.264	2.890
2			-0.635	-0.468	-1.508	0.812	1.284	1.580	-1.800	0.748
3				-1.044	0.536	-2.092	-0.774	-3.314	-0.348	0.272
4					0.011	2.918	0.698	0.752	-3.496	0.460
5						-0.987	3.940	2.926	-0.508	1.648
6							0.198	-0.362	-2.402	1.646
7								-1.732	-1.334	-3.070
8									0.860	0.648
9										0.305

The system to be identified was a second-order Volterra system with 10 linear coefficients and 55 quadratic coefficients. The linear and quadratic coefficients of the nonlinear system are given in Table IV. The adaptive filter had the same number of coefficients as the unknown system. The input signal to the unknown system was obtained by filtering pseudorandom white Gaussian noise with zero mean and variance 0.0248 by an FIR filter with impulse response $h(n)$ given by

$$h(n) = \begin{cases} 0.9045, & n = 1 \\ 1.0, & n = 1 \\ 0.9045, & n = 2 \\ 0, & otherwise. \end{cases} \tag{77}$$

The parameters of the input signal were chosen such that the output of the unknown system had approximately unit variance. The desired signal $d(n)$ was obtained by adding zero-mean white Guassian noise to the output of the unknown system. The measurement noise was uncorrelated with the input signal. Several experiments were conducted using two output signal-to-measurement noise ratios of 20 dB and 30 dB and two weighting factors 0.995 and 0.9975. Such signal-to-noise ratios are fairly common in such applications as decision feedback equalization of telephone lines and echo cancellation. The results presented are ensemble averages of 50 independent runs of 5000 samples each. Performance evaluations were carried out by plotting the mean squared a posteriori error and the squared norms of the linear and quadratic coefficient error vectors.

The following procedure, details of which can be found in [1], was adopted for computing the coefficients of the filter.

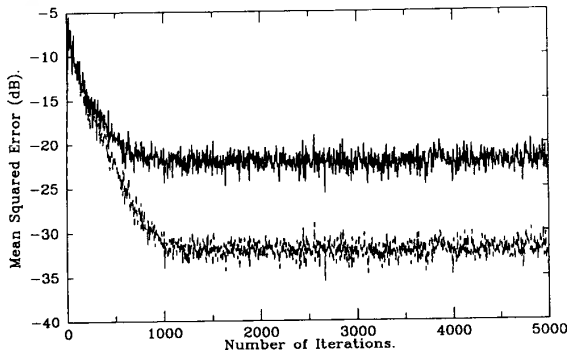


Fig. 1. Learning curves for the QR-RLS sequential processing algorithm used for adaptive Volterra filtering. Weighting factor=0.995. Number of bits for integer part=5. Number of bits for fractional part=16. Solid curve: 20-dB measurement noise. Dashed curve: 30-dB measurement noise.

Define the orthogonal matrix $P_L(n)$ such that

$$P_L(n)\begin{bmatrix}\underline{\boldsymbol{b}}(n)\\ 1\end{bmatrix}=\begin{bmatrix}\mathbf{0}\\ \beta_L(n)\end{bmatrix} \tag{78}$$

where

$$\underline{\boldsymbol{b}}(n)=\frac{R_L^{-\mathrm{T}}(n-1)\tilde{x}(n)}{\sqrt{\lambda}} \tag{79}$$

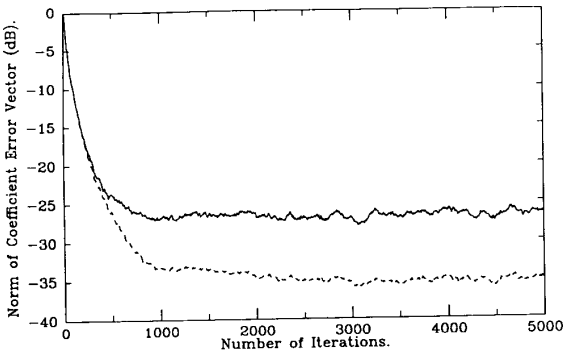
and $\tilde{x}(n)$ is the input data vector at time n. $P_L(n)$ is a Givens rotation matrix that successively annihilates the elements of the $L\times 1$ vector $\underline{\boldsymbol{b}}(n)$, starting from the top, by rotating them into the element at the bottom. One can show that [1]

$$P_L(n)\begin{bmatrix}R_L^{-\mathrm{T}}(n-1)/\sqrt{\lambda}\\ \mathbf{0}^{\mathrm{T}}\end{bmatrix}=\begin{bmatrix}R_L^{-\mathrm{T}}(n)\\ \tilde{g}^{\mathrm{T}}(n)\end{bmatrix} \tag{80}$$

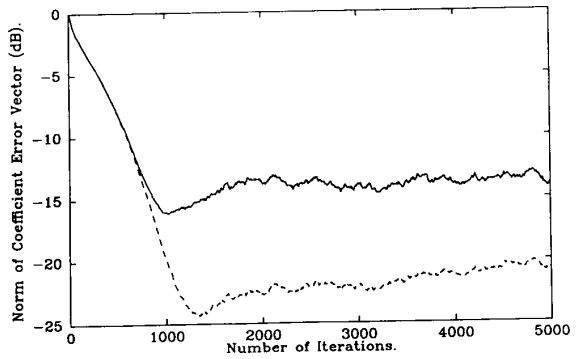
and that the coefficient vector can be recursively updated as

$$\underline{\boldsymbol{W}}(n)=\underline{\boldsymbol{W}}(n-1)+\frac{\tilde{g}(n)}{\beta_L(n)}\frac{\alpha(n)}{\gamma_L(n)} \tag{81}$$

where $\alpha(n)$ is the a priori joint process estimation error and $\gamma_L(n)$ is as defined earlier.



(a)



(b)

Fig. 2. Norm of coefficient error vector for the QR-RLS adaptive Volterra filter with sequential processing. Weighting factor=0.995. Number of bits for integer part=5. Number of bits for fractional part=16. Solid curve: 20-dB measurement noise. Dashed curve: 30-dB measurement noise. (a) Norm of the linear coefficient error vector. (b) Norm of the quadratic coefficient error vector.

In Table V we have presented the values of the mean squared difference between the joint process estimation error computed using the maximum precision available on the computer and that obtained using a given precision for the

TABLE V
STEADY-STATE MEAN-SQUARED VALUES OF THE NUMERICAL ERRORS FOR DIFFERENT ORDERS OF THE QR-RLS VOLTERRA FILTER WITH SEQUENTIAL PROCESSING

Int.-Frac. Bits	Order 1	Order 5	Order 10
$\lambda = 0.995, \quad SNR = 20dB$			
5-15	8.23×10^{-9}	2.62×10^{-6}	1.38×10^{-5}
5-16	2.18×10^{-9}	5.47×10^{-7}	2.40×10^{-6}
5-18	7.78×10^{-10}	3.06×10^{-8}	9.66×10^{-8}
$\lambda = 0.9975, \quad SNR = 20dB$			
6-15	2.21×10^{-8}	6.23×10^{-6}	4.56×10^{-5}
6-16	3.83×10^{-9}	1.04×10^{-6}	6.42×10^{-6}
6-18	3.96×10^{-10}	3.97×10^{-8}	1.89×10^{-7}
$\lambda = 0.995, \quad SNR = 30dB$			
5-15	8.38×10^{-9}	2.62×10^{-6}	1.36×10^{-5}
5-16	2.15×10^{-9}	5.45×10^{-7}	2.35×10^{-6}
5-18	7.76×10^{-10}	3.05×10^{-8}	9.40×10^{-8}
$\lambda = 0.9975, \quad SNR = 30dB$			
6-15	2.21×10^{-8}	6.22×10^{-6}	4.53×10^{-5}
6-16	3.77×10^{-9}	1.04×10^{-6}	6.38×10^{-6}
6-18	4.01×10^{-10}	3.97×10^{-8}	1.86×10^{-7}

TABLE VI
VALUES OF THE TIME-AVERAGED MEAN-SQUARED DIFFERENCE BETWEEN THE "INFINITE" PRECISION IMPLEMENTATION AND THE IMPLEMENTATION WITH A FIXED NUMBER OF BITS FOR BOTH THE INTEGER AND FRACTIONAL PARTS FOR THE SEQUENTIAL ALGORITHM

Int.-Frac. Bits	Mean-Squared Numerical Error	
	$\lambda = 0.995, \quad SNR = 20dB$	$\lambda = 0.995, \quad SNR = 30dB$
5-15	1.38×10^{-5}	1.36×10^{-5}
5-16	2.40×10^{-6}	2.35×10^{-6}
5-18	9.66×10^{-8}	9.40×10^{-8}
	$\lambda = 0.9975, \quad SNR = 20dB$	$\lambda = 0.9975, \quad SNR = 30dB$
6-15	4.56×10^{-5}	4.53×10^{-5}
6-16	6.42×10^{-6}	6.38×10^{-6}
6-18	1.89×10^{-7}	1.86×10^{-7}

The minimum number of bits for the integer part was 5 for $\lambda = 0.995$ and 6 for $\lambda = 0.9975$. It was observed that when the number of bits for the fractional part is less than 12 bits the learning curve converges to a value that is greater than the variance of the measurement noise by a factor greater than 2. However, when the number of bits for the fractional part is greater than 12 the learning curve converges to a value that is only slightly greater than the variance of the measurement noise. Comparing the performance of the systems with different model orders we observe that the numerical errors increase as the order of the Volterra filter increases. Thus, the QR-RLS Volterra filter seems to share this property that is true for most lattice filters. One way to mitigate this problem in lattice implementations is to scale the variables in each stage differently so that the dynamic ranges of the variables are about the same in all the stages. It may be possible to develop a similar strategy for the QR-RLS filters also. However, we have not investigated this possibility.

IV. CONCLUSIONS

In this paper we presented a QR-RLS adaptive algorithm for second-order Volterra filtering. The algorithm is based solely on Givens rotation. While no theoretical proof was presented, a large number of simulation experiments that were conducted using fixed point implementation of the adaptive filter appear to indicate that the algorithm is numerically stable, at least in the configuration employed in the simulations. It is also highly amenable to parallel implementations using systolic arrays. The algorithm processes the channels individually and is a sequential processing algorithm. Sequential processing leads to a scalar implementation. The computational complexity of the algorithm is comparable to that of the other fast RLS Volterra filters that are available in the literature. Another attractive feature of the sequential algorithm is that since the processing is the same as in a single-channel algorithm, knowledge of the single-channel algorithm can be applied to the multichannel case. It is hoped that the availability of computationally efficient and numerically stable algorithms for adaptive nonlinear filtering will spur the practical applications of such filters.

REFERENCES

[1] S. T. Alexander and A. L. Ghirnikar, "A method for recursive least squares filtering based upon an inverse QR decomposition," *IEEE Trans. Signal Proc.*, vol. 41, no. 1, pp. 20–30, Jan. 1993.

[2] M. G. Bellanger, "The FLS-QR algorithm for adaptive filtering," *Signal Proc.*, vol. 17, pp. 291–304, 1989.

[3] M. G. Bellanger and P. A. Regalia, "The FLS-QR algorithm for adaptive filtering: The case of multichannel signals," *Signal Proc.*, vol. 22, no. 2, pp. 291–304, Feb. 1991.

[4] J. M. Cioffi, "The fast adaptive rotors RLS algorithm," *IEEE Trans. Acoust., Speech, Signal Proc.*, vol. 38, no. 4, pp. 631–653, April 1990.

[5] G. Golub, and C. F. Van Loan, *Matrix Computations.* Baltimore. Johns-Hopkins University Press, 1983.

[6] S. Haykin, *Adaptive Filter Theory.* Englewood Cliffs, NJ: Prentice-Hall, 1985.

[7] T. Koh and E. J. Powers, "Second order Volterra filtering and its applications to nonlinear system identification," *IEEE Trans. Acoustics, Speech, Signal Proc.*, vol. ASSP-33, no. 6, pp. 1445–1455, Dec. 1985.

[8] J. Lee and V. J. Mathews, "A fast recursive least-squares adaptive second order Volterra filter and its performance analysis," *IEEE Trans. Signal Proc.*, vol. 41, no. 3, Mar. 1993.

[9] H. Lev-Ari, "Modular architectures for adaptive multichannel lattice algorithms," *IEEE Trans. Acoust., Speech, Signal Proc.*, vol. ASSP-35, no. 4, pp. 543–552, April 1987.

[10] P. Lewis, Algorithms and architectures for adaptive least squares signal processing, with applications in magnetoencephalography. Ph.D. Thesis, Univ. of Southern California, 1988.

[11] F. Ling, "Givens rotation based least-squares lattice and related algorithms," *IEEE Trans. Signal Proc.*, vol. 39, no. 7, pp. 1541–1551, July 1991.

[12] V. J. Mathews, "Adaptive polynomial filters," *IEEE Signal Proc. Mag.*, pp. 10–26, July 1991.

[13] I. K. Proudler, J. G. McWhirter, and T. J. Shepherd, "Fast QRD-based algorithms for least squares linear prediction," in *Proc. IMA Mathematics in Signal Processing Conf.* (Warwick, England, 1988).

[14] P. A. Regalia and M. G. Bellanger, "On the duality between fast QR methods and lattice methods in least-squares adaptive filtering," *IEEE Trans. Signal Proc.*, vol. 39, pp. 879–891, April 1991.

[15] W. J. Rugh, *Nonlinear System Theory.* Baltimore: Johns Hopkins Univ. Press, 1981.

[16] M. Schetzen, *The Volterra and Wiener Theory of the Nonlinear Systems,* New York: Wiley, 1980.

[17] G. L. Sicuranza, "Quadratic filters for signal processing," *Proc. IEEE,* vol. 80, no. 8, pp. 1263–1285, Aug. 1992.

[18] D. T. M. Slock, L. Chisci, H. Lev-Ari, and T. Kailath, "Modular and numerically stable fast transversal filters for multichannel multiexperiment RLS," *IEEE Trans. Signal Proc.*, vol. 40, no. 4, pp. 784–802, April 1992.

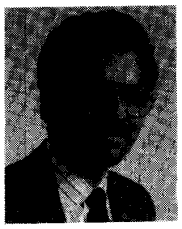
[19] M. A. Syed, "QRD-based fast RLS multichannel adaptive algorithms," in *IEEE Int. Conf. Acoustics, Speech and Signal Processing 91* (Toronto, Canada, 1991), pp. 1837–1840.

[20] M. A. Syed and V. John Mathews, "Lattice and QR decomposition-based algorithms for recursive least squares adaptive nonlinear filters," in *Proc. Int. Symp. Circuits and Systems* (New Orleans, LA, May 1990), pp. 262–265.

[21] B. Yang and J. F. Bohme, "Rotation based RLS algorithms: Unified derivations, numerical properties and parallel implementations," *IEEE Trans. Signal Proc.*, vol. 40, no. 5, pp. 1151–1167, May 1992.

Mushtaq A. Syed was born in Poona, India. He received the B.E. degree in instrumentation and control from the University of Poona, the M.Sc. degree in biomedical engineering from the University of Saskatchewan, Canada, and the M.E. and Ph.D. degree in electrical engineering from the University of Utah.

At present he is working as a DSP engineer with Digicom Systems, Inc., of Milpitas, Calif. His research interests include nonlinear adaptive signal processing, image processing, and digital communications.



V. John Mathews (S'82—M'84—SM'90) was born in Nedungadappally, Kerala, India, in 1958. He received the B.E. (Hons.) degree in electronics and communication engineering from the University of Madras, India, in 1980, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Iowa, Iowa City, in 1981 and 1984, respectively.

From 1980 to 1984 he held a Teaching Research Fellowship at the University of Iowa, where he also worked as a Visiting Assistant Professor with the Department of Electrical and Computer Engineering from 1984 to 1985. Since 1985 he has been with the Department of Electrical Engineering, University of Utah, Salt Lake City, where he is now an Associate Professor. His research interests include adaptive filtering, spectrum estimation, and data compression.

Dr. Mathews is a past Associate Editor of the *IEEE Transactions on Signal Processing* and is currently an Associate Editor of the *IEEE Signal Processing Letters*. He is a member of the Digital Signal Processing Technical Committee of the IEEE Signal Processing Society.